Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №4

Название: Параллельное умножение матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) А.О.Найденышев (И.О. Фамилия)

Преподователь _____ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Цель работы: изучение возможности параллельных вычислений и использование такого подхода на практике.

В ходе лабораторной работы требуется:

1) выбрать алгоритм для рассмотрения;
2) описать обычную версию;
3) выполнить 2 параллельные версии;
4) запустить эксперименты на каждый при различном числе потоков 1,2,4,8 ...4M, где M - количество логических ядер на компьютере;
5) проверить, всегда ли при росте потоков, время работы снижается;
6) описать главный и рабочий поток схемой;
7) описать точки сборки.

В данной лабораторной работе был выбран алгоритм Винограда. Необходимо сравнить зависимость времени работы алгоритма от числа параллельных потоков и размера матриц, провести сравнение стандартного и параллельного алгоритмов.

1 Аналитический раздел

В данном разделе представлено описание стандартного и параллельного алгоритмов Винограда.

1.0.1 Общие сведения

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

1.0.2 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. [1]

Рассмотрим два вектора $V = (v_1,v_2,v_3,v_4)$ и $W = (w_1,w_2,w_3,w_4)$. Их скалярное произведение равно:

$$V * W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4 \tag{1.1}$$

Это равенство можно переписать в виде:

$$V * W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4 \tag{1.2}$$

Менее очевидно, что выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. На практике это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.0.3 Параллельное программирование

Параллельное программирование. Параллельное программирование служит для создания программ, эффективно использующих вычислительные ресурсы за счет одновременного исполнения кода на нескольких вычислительных узлах. Для создания параллельных приложений используются параллельные языки программирования и специализированные системы поддержки параллельного программирования, такие как MPI и OpenMP. Параллельное программирование является более сложным по сравнению с последовательным как в написании

кода, так и в его отладки. Для облегчения процесса параллельного программирования существуют специализированные инструменты, например, отладчик TotalView[2].

1.0.4 Параллельный алгоритм Винограда

Для того, чтобы добиться большей эффективности алгоритма умножения матриц Винограда, необходимо расспараллелить ту часть алгоритма, которая содержит больше всего вложенных циклов. Вычисление результата для каждой строки не зависит от результата умножения для других строк. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.

Для проверки эффективности выбранного метода, также можно распараллелить циклы двойной вложенности в начале алгоритма (циклы вычисления векторов mulH и mulV).

1.0.5 Вывод

Было представлено математическое описание стандартного и параллельного алгоритмов Винограда. Основная идея заключается в том, чтобы распараллелить главный цикл тройной вложенности.

2 Констукторский раздел

В данном разделе представлены схемы разработанных алгоритмов. Также описывается часть алгоритма, которая будет распараллеливаться..

2.1 Разработка алгоритма

Ниже изображена схема стандартного алгоритма умножения матриц.(рисунок 2.1)

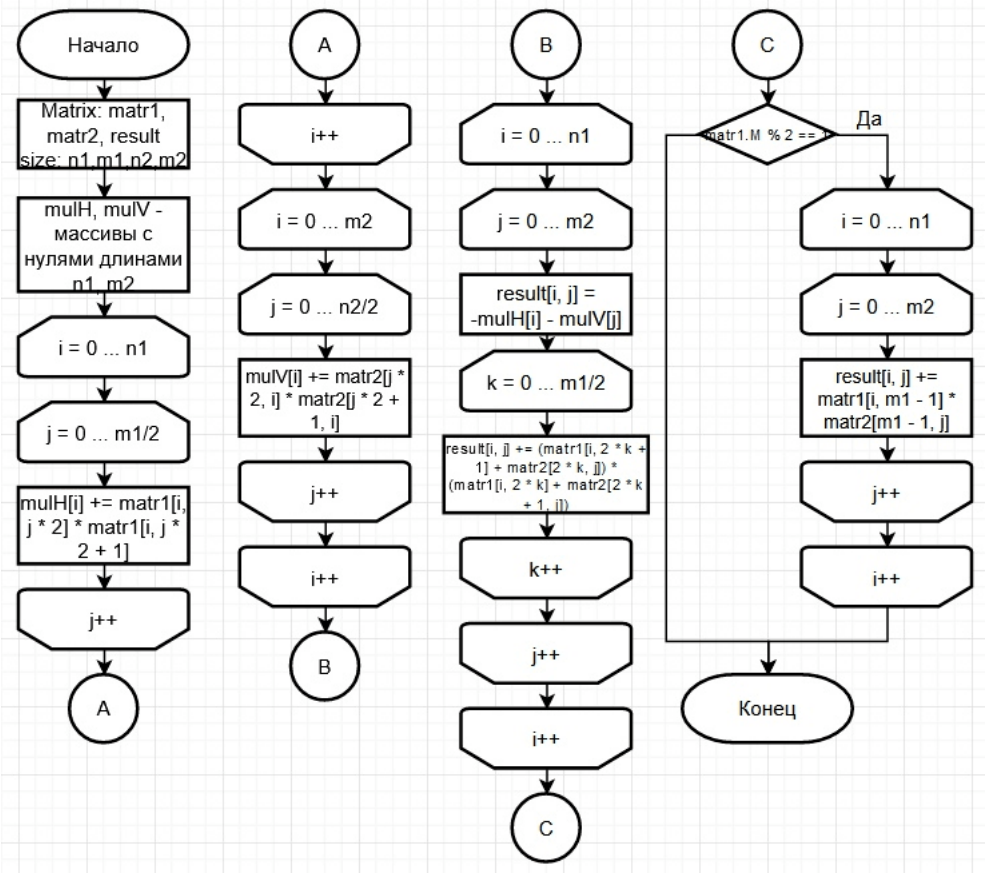

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

2.2 Распараллеливание программы

В представленном алгоритме будет распараллеливаться цикл тройной вложенности (участок между B и C). Это должно ускорить время работы алгоритма.

Также для проверки эффективности будут распараллеливаться два первых цикла отдельно. Будут построены графики для данной реализации и сравниться время работы алгоритмов.

В предложенном алгоритме данным участком будет являться основной двойной цикл вычислений. Данный блок программы как раз предлагается распараллелить.

2.3 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения:

1) обеспечить ввод размера матриц;

2) обеспечить вывод результирующей матрицы тремы способами (стандартное умножение, параллельное и оптимизированное параараллельное);

2.4 Методы тестирования

Тестирование ПО будет проводиться методом черного ящика.

3 Технологический раздел

В данном разделе даны общие требования к программе, средства реализации и сама реализация алгоритмов.

3.1 Средства реализации

В данной работе используется язык программирования C# [3], так как он позволяет написать программу в относительно малый срок. В качестве среды разработки использовалась Visual Studio 2017 [4]. Для замера процессорного времени был использован класс Stopwatch [5].

Многопоточное программирование было реализовано с помощью пространства имен System.Threading [6].

В листинге 1 реализован стандартный алгоритм Винограда.

Листинг 3.1 — Стандартный алгоритм Винограда

```
1     public static int[][] Vinograd(int[][] matr1, int[][] matr2)
2         {
3             int row1 = matr1.Length;
4             int row2 = matr2.Length;
5
6             if (row1 == 0 || row2 == 0)
7                 return null;
8
9             int col1 = matr1[0].Length;
10            int col2 = matr2[0].Length;
11
12            if (col1 != row2)
13                return null;
14
15            int[] mulH = new int[row1];
16            int[] mulV = new int[col2];
17
18            int[][] res = new int[row1][];
19            for (int i = 0; i < row1; i++)
20                res[i] = new int[col2];
21
22            for (int i = 0; i < row1; i++)
23            {
24                for (int j = 0; j < col1 / 2; j++)
25                {
26                    mulH[i] = mulH[i] + matr1[i][j * 2] * matr1[i][j * 2 + 1];
27                }
28            }
29
30            for (int i = 0; i < col2; i++)
31            {
```

```
32                  for (int j = 0; j < row2 / 2; j++)
33                  {
34                      mulV[i] = mulV[i] + matr2[j * 2][i] * matr2[j * 2 + 1][i];
35                  }
36              }
37
38              for (int i = 0; i < row1; i++)
39              {
40                  for (int j = 0; j < col2; j++)
41                  {
42                      res[i][j] = -mulH[i] - mulV[j];
43                      for (int k = 0; k < col1 / 2; k++)
44                      {
45                          res[i][j] = res[i][j] + (matr1[i][2 * k + 1] + matr2[2 *
                                k][j]) * (matr1[i][2 * k] + matr2[2 * k + 1][j]);
46                      }
47                  }
48              }
49
50              if (col1 % 2 == 1)
51              {
52                  for (int i = 0; i < row1; i++)
53                  {
54                      for (int j = 0; j < col2; j++)
55                      {
56                          res[i][j] = res[i][j] + matr1[i][col1 - 1] * matr2[col1 -
                                1][j];
57                      }
58                  }
59              }
60
61              return res;
62          }
```

В листинге 2 реализован параллельный алгоритм Винограда

Листинг 3.2 — Параллельный алгоритм Винограда

```
1           public static int[][] ParallelMultVin(int[][] matr1, int[][] matr2, int
                nThreads, int flag)
2           {
3               Thread[] threadsArray = new Thread[nThreads];
4
5               int row1 = matr1.Length;
6               int row2 = matr2.Length;
7
8               if (row1 == 0 || row2 == 0)
9                   return null;
```

```
10
11              int col1 = matr1[0].Length;
12              int col2 = matr2[0].Length;
13
14              if (col1 != row2)
15                  return null;
16
17              int[] mulH = new int[row1];
18              int[] mulV = new int[col2];
19
20              int[][] res = new int[row1][];
21              for (int i = 0; i < row1; i++)
22                  res[i] = new int[col2];
23
24              for (int i = 0; i < row1; i++)
25              {
26                  for (int j = 0; j < col1 / 2; j++)
27                  {
28                      mulH[i] = mulH[i] + matr1[i][j * 2] * matr1[i][j * 2 + 1];
29                  }
30              }
31
32              for (int i = 0; i < col2; i++)
33              {
34                  for (int j = 0; j < row2 / 2; j++)
35                  {
36                      mulV[i] = mulV[i] + matr2[j * 2][i] * matr2[j * 2 + 1][i];
37                  }
38              }
39
40              int rowsForThread = row1 / nThreads;
41              int start = 0;
42              for (int i = 0; i < nThreads; i++)
43              {
44                  int end = start + rowsForThread;
45
46                  if (i == nThreads - 1)
47                      end = row1;
48
49                  MatrixForParall p = new MatrixForParall(res, matr1, matr2, mulV,
                        mulH, start, end, col1, col2);
50
51                  if (flag == 1)
52                      threadsArray[i] = new Thread(new
                            ParameterizedThreadStart(MatrixForParall.MainCycleOptimize));
53                  else
```

```
54                      threadsArray[i] = new Thread(new
                            ParameterizedThreadStart(MatrixForParall.MainCycle));
55                  threadsArray[i].Start(p);
56
57                  start = end;
58              }
59              foreach (Thread thread in threadsArray)
60              {
61                  thread.Join();
62              }
63
64
65              if (col1 % 2 == 1)
66              {
67                  for (int i = 0; i < row1; i++)
68                  {
69                      for (int j = 0; j < col2; j++)
70                      {
71                          res[i][j] = res[i][j] + matr1[i][col1 - 1] * matr2[col1 -
                                1][j];
72                      }
73                  }
74              }
75
76              return res;
77          }
```

В листинге 3 реализован алгоритм Винограда с распараллеленым алгритмом Винограда

Листинг 3.3 — Распараллеленный главный цикл

```
1       class MatrixForParall
2       {
3           public int[][] res, matr1, matr2;
4           public int[] mulV, mulH;
5           public int start, end, col1, col2;
6
7           public MatrixForParall(int[][] res, int[][] matr1, int[][] matr2, int[]
               mulV, int[] mulH, int start, int end, int col1, int col2)
8           {
9               this.res = res;
10              this.matr1 = matr1;
11              this.matr2 = matr2;
12              this.mulV = mulV;
13              this.mulH = mulH;
14              this.start = start;
15              this.end = end;
16              this.col1 = col1;
```

```
17              this.col2 = col2;
18          }
19
20          public static void MainCycle(object obj)
21          {
22              MatrixForParall p = (MatrixForParall)obj;
23
24              for (int i = p.start; i < p.end; i++)
25              {
26                  for (int j = 0; j < p.col2; j++)
27                  {
28                      p.res[i][j] = -p.mulH[i] - p.mulV[j];
29                      for (int k = 0; k < p.col1 / 2; k++)
30                      {
31                          p.res[i][j] = p.res[i][j] + (p.matr1[i][2 * k + 1] +
                                p.matr2[2 * k][j]) * (p.matr1[i][2 * k] + p.matr2[2 * k +
                                1][j]);
32                      }
33                  }
34              }
35          }
36
37          public static void MainCycleOptimize(object obj)
38          {
39              MatrixForParall p = (MatrixForParall)obj;
40              int[][] res = p.res, matr1 = p.matr1, matr2 = p.matr2;
41              int[] mulH = p.mulH, mulV = p.mulV;
42              int start = p.start, end = p.end;
43              int col2 = p.col2, col1 = p.col1;
44
45              for (int i = start; i < end; i++)
46              {
47                  for (int j = 0; j < col2; j++)
48                  {
49                      res[i][j] = -mulH[i] - mulV[j];
50                      for (int k = 0; k < col1 / 2; k++)
51                      {
52                          res[i][j] = res[i][j] + (matr1[i][2 * k + 1] + matr2[2 *
                                k][j]) * (matr1[i][2 * k] + matr2[2 * k + 1][j]);
53                      }
54                  }
55              }
56          }
57      }
```

3.1.1 Вывод

В данном разделе были даны общие требования к программе, описаны средства реализации, были представлены сведения о модулях программы, а также реализованы алгоритмы умножения матриц (Винограда и параллельный Винограда).

4 Экспериментальный раздел

В данном разделе представлены результаты работы программы и приведен анализ времени работы алгоритмов.

4.1 Анализ времени работы алгоритмов

Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i5-8250U CPU [7], 8 логических процессора, под управлением Windows 10 с 8 Гб оперативной памяти.

Ниже представлен график зависимости времени работы алгоритмов и количества потоков (рисунок 4.1).

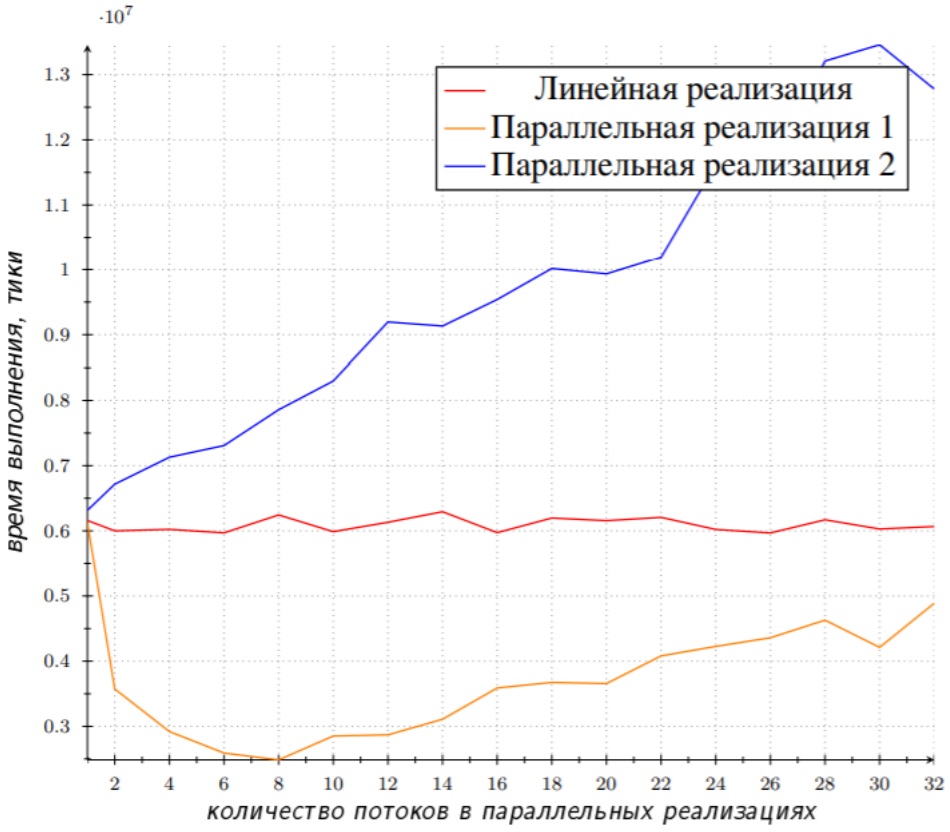

Рисунок 4.1 — Результаты замеров процессорного времени в первом эксперименте.

Для передачи в распараллеленную функцию данных в C# необходимо передавать данные только в объекте, изначально создается объект, который хранит необходимые данные, и в функцие данные переприсваиваются переменным. Из-за этого время работы 2 параллельной реализации будет намного больше линейной реализации и 1 параллельной реализации.

Ниже представлен график зависимости времени работы алгоритмов и размера матриц.

Из-за того, что замеры проводятся с 8 логическими процессорами - эффективнее всего использовать 8 потоков.(рисунок 4.2)

В ходе экспериментов по замеру времени работы было установлено:

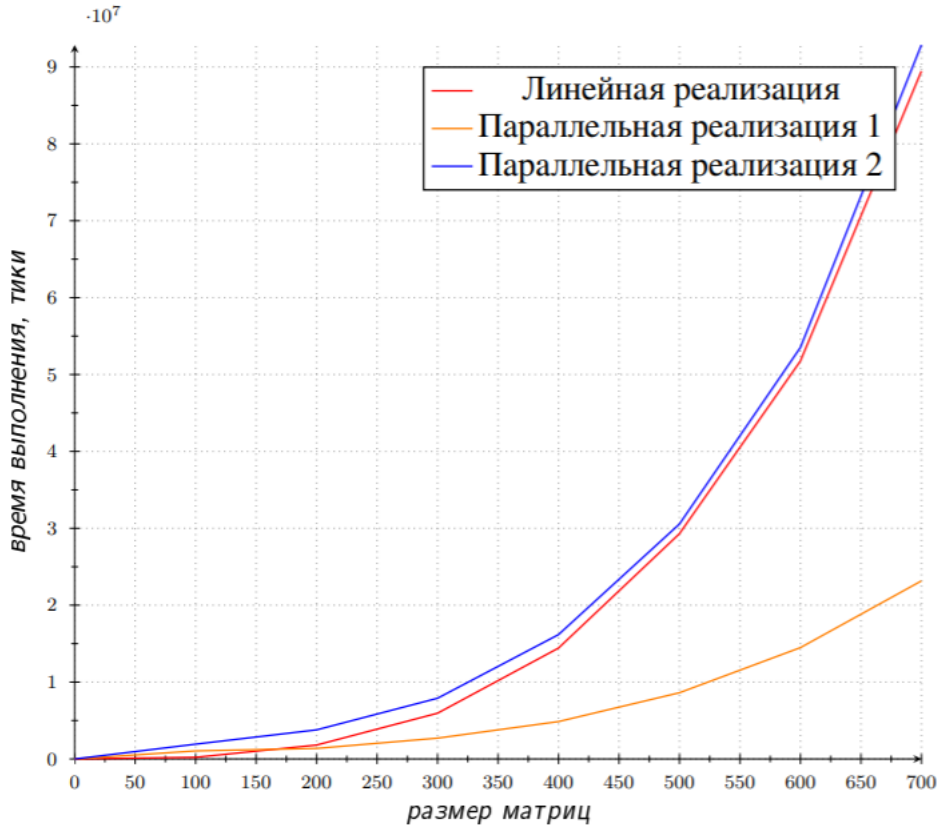

Рисунок 4.2 — Результаты замеров процессорного времени во втором эксперименте.

1) с увеличением количества потоков увеличивается время работы параллельной реализации 2. Линенйая реализация показывает одинаковый результат. А параллельная реализация 2 показывает, что при 8 потоках - она наиболее эффективна, а с увеличением увеличивается время;

2) при одинаковом количестве потоков и разных размерах матриц, самым эффективной является вторая параллельная реализация, а первая и линейная работают идентично.

Можно сделать вывод, что самой эффективной реализацией является вторая и, что следует распараллеливать главный цикл (состоящий из 3 вложенных циклов).

Заключение

В ходе выполнения лабораторной работы были изучены возможности параллельных вычислений и был использован такой подход на практике. Были даны теоритически оценки алгоритмов умножения матриц. Также сравнили время работы алгоритмов, в результате которого стало понятно, что самой эффективной реализацией была та, когда было необходимо распараллелить главный цикл алгоритма Винограда. Данная реализация быстрее линейной примерно в 2.5 раза.

Список использованных источников

1. Умножение матриц. // [Электронный ресурс]. Режим доступа: http://www.algolib.narod.ru/Math/Matrix.html, (дата обращения: 4.11.2020).

2. Параллельное программирование. // [Электронный ресурс]. Режим доступа: https://www.viva64.com/ru/t/0038/, (дата обращения: 4.11.2020).

3. C#. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/, (дата обращения: 4.11.2020).

4. IDE Visual Studio 2019. // [Электронный ресурс]. Режим доступа: https://visualstudio.microsoft.com/ru/vs/, (дата обращения: 4.11.2020).

5. Stopwatch. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=netcore-3.1, (дата обращения: 4.11.2020).

6. Thread. // [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.thread?view=netcore-3.1, (дата обращения: 4.11.2020).

7. Intel® Core™ i3-8130U Processor. // [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/137977/intel-core-i3-8130u-processor-4m-cache-up-to-3-40-ghz.html, (дата обращения: 4.11.2020).